Composing Reliable Systems with Virtualization and the Yocto Project®

Tim Orling, Intel Corporation
Christopher Clark, OpenXT Project

Yocto Project Summit *Virtual*, Europe, 2020

Agenda

- **Why Virtualization?**
- **Xen Hypervisor**
  - Xen on Raspberry Pi
- **KVM Hypervisor**
  - Unmodified Ubuntu cloudimage on NUC

Content and Continuous Integration

https://gitlab.com/moto-timo/yp-dev-day_virtualization.git

Why Virtualization?

The vendor kernel is dead.

Long live the secure vendored kernels.

Why Virtualization?

- **Use the latest stable kernel**
  - This is the best kernel we have (excluding Linux distribution kernels)
  - Security fixes
  - Bug fixes
  - New features
  - Latest drivers
- **NOTE:** this is why every vendor should be upstreaming first

Why Virtualization?

- **Use the latest stable kernel**
  - This is the best kernel we have (excluding Linux distribution kernels)
  - Security fixes
  - Bug fixes
  - New features
  - Latest drivers
- **NOTE:** this is why every vendor should be upstreaming first

Why Virtualization?

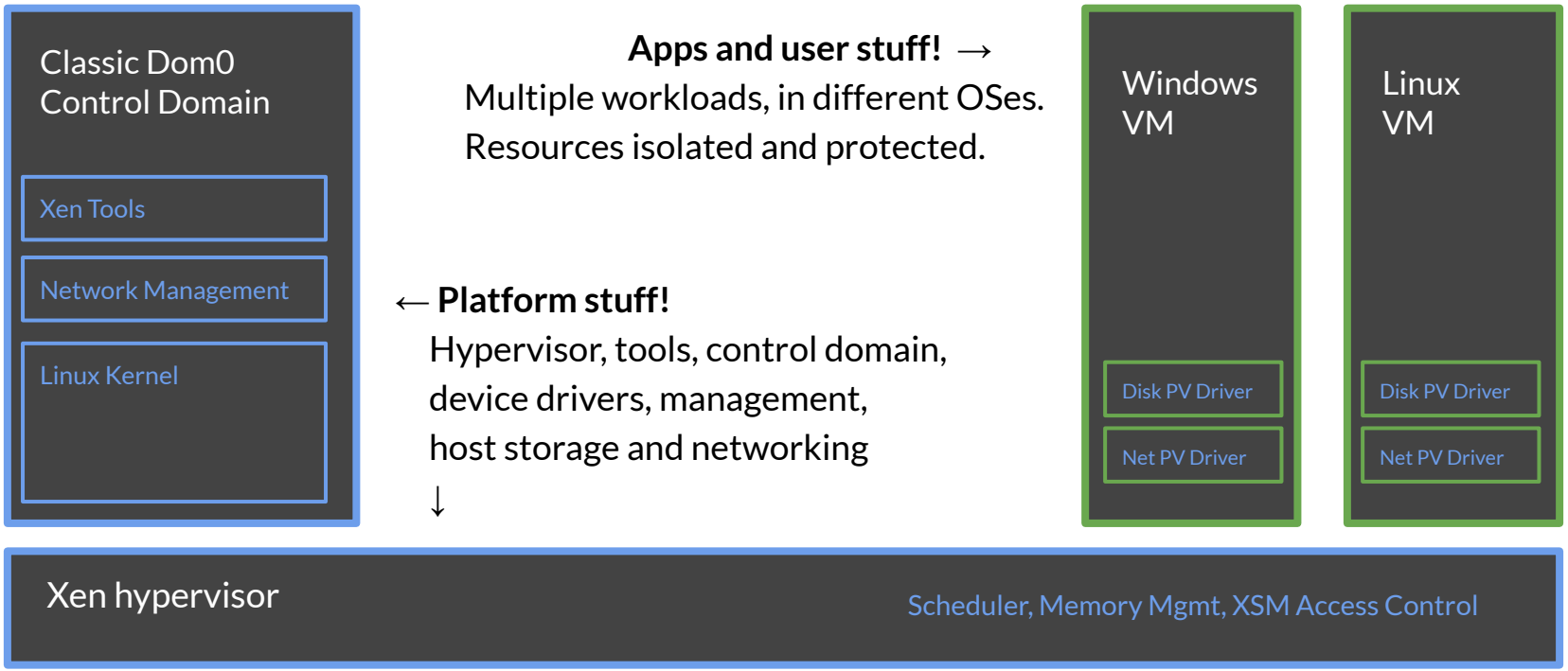
- **Ship images with complete control**
  - Contents
  - Versions
  - Licenses
- **Run applications in an isolated environment**
- **Run vendor kernels and vendor images**

Xen Hypervisor

Christopher Clark, October 2020

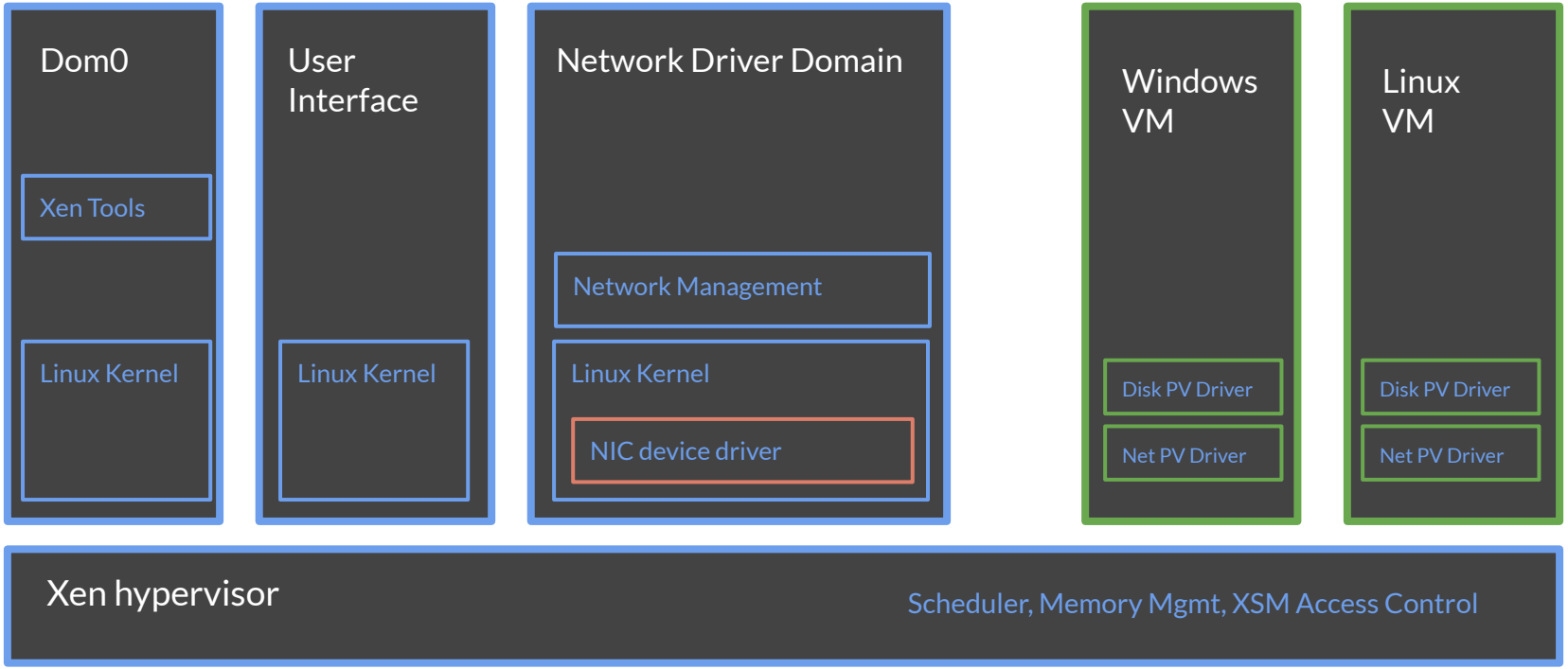
Xen Hypervisor: A brief look

Powerful. Flexible. Open.

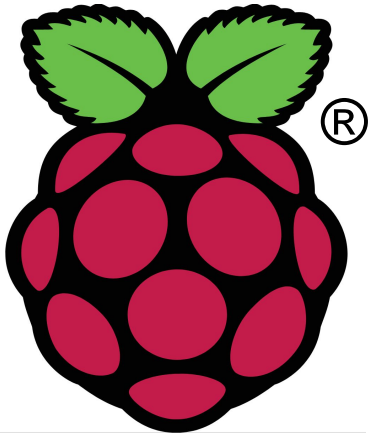


Xen Hypervisor: A brief look

Powerful. Flexible. Open.



Yes, but can I run it on my



?

hmm, let’s see about that...

Bitbaking Xen for the Raspberry Pi 4

Our **essential basic ingredients** are in layers:

- Xen support is in **meta-virtualization** (source, list)
- Raspberry Pi 4 support is in **meta-raspberrypi** (source)
- We'll use **poky** with **meta-openembedded**

Add in a *fresh zesty sprinkle* of:

- a **brand-new Xen-on-Raspberry Pi 4 patch series for meta-virtualization**

… along with some **classic Yocto spiciness**:

- configure your local.conf:

```
MACHINE=”raspberrypi4-64"
DISTRO_FEATURES_append = " virtualization xen";
QEMU_TARGETS = "i386 x86_64 aarch64 arm"
```

"Bake!" : **`bitbake xen-image-minimal`** → SD-card image!

Xen for the Raspberry Pi 4 : SD-card

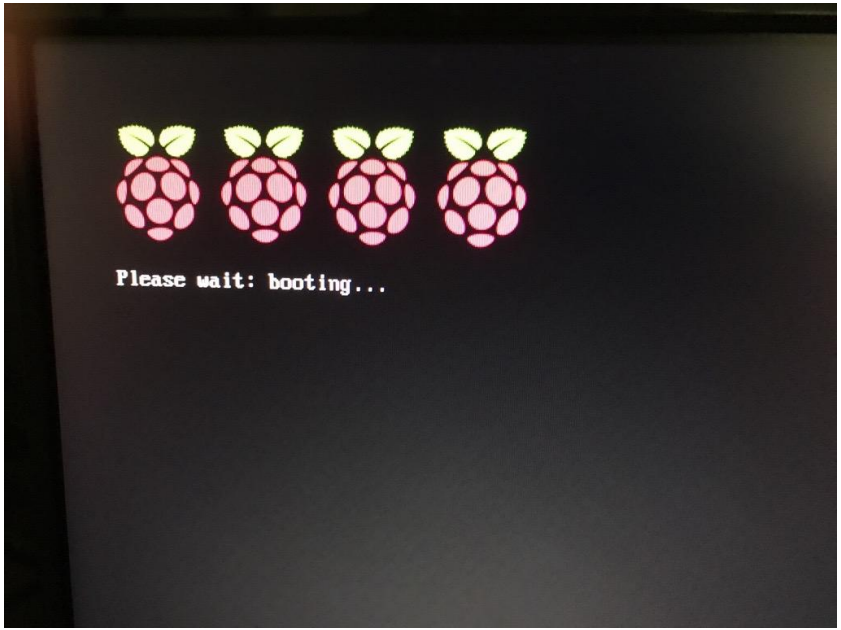
The card image contains two partitions:

- **boot**, which includes:
  - xen : hypervisor binary
  - Image : Linux kernel binary
  - bcm2711-rpi-4-b.dtb : Device tree for the Raspberry Pi 4
  - overlays : Device tree overlays
  - boot.scr : Xen-specific u-boot launch script
  - config.txt : Raspberry Pi configuration settings
  - ...

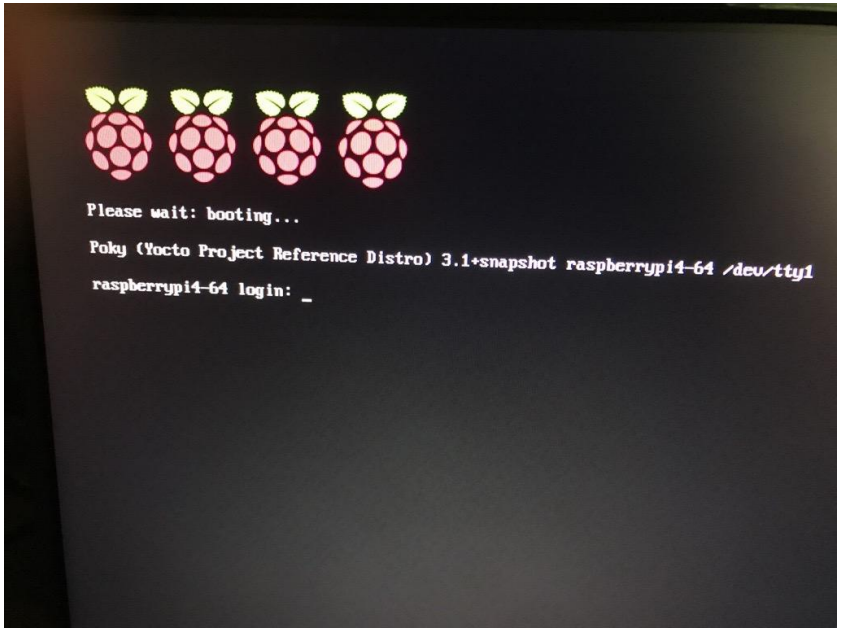
- **root** filesystem for Domain 0
  - poky Linux filesystem
  - contains the familiar Xen tools
  - *does not contain* the hypervisor or the dom0 Linux kernel

Xen Hypervisor on Raspberry Pi 4

Insert the SD card, fire up the power, … stand well back...

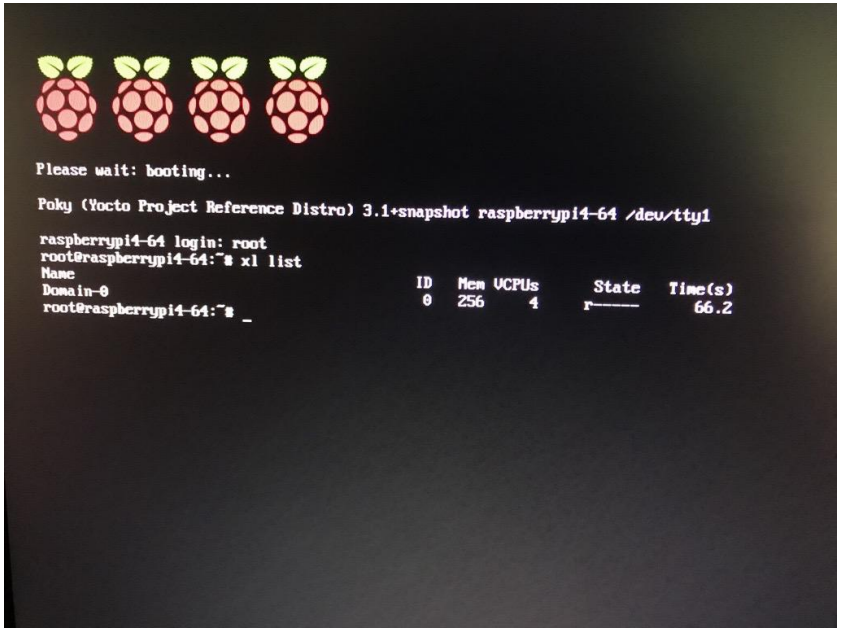


Xen Hypervisor on Raspberry Pi 4



Hooray!

Xen Hypervisor on Raspberry Pi 4



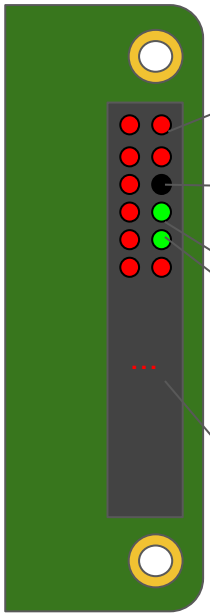
ok, proof that it is actually there and working

Xen Hypervisor on Raspberry Pi 4: serial

To obtain for yourself a luxury device such as this, the terms you can use to proceed are: “pl2303hx USB serial”

Xen Hypervisor on Raspberry Pi 4: serial

OK - time to wire it up! There are three wires to connect:



Pins indicated red here: *nope, do not need those for this.*

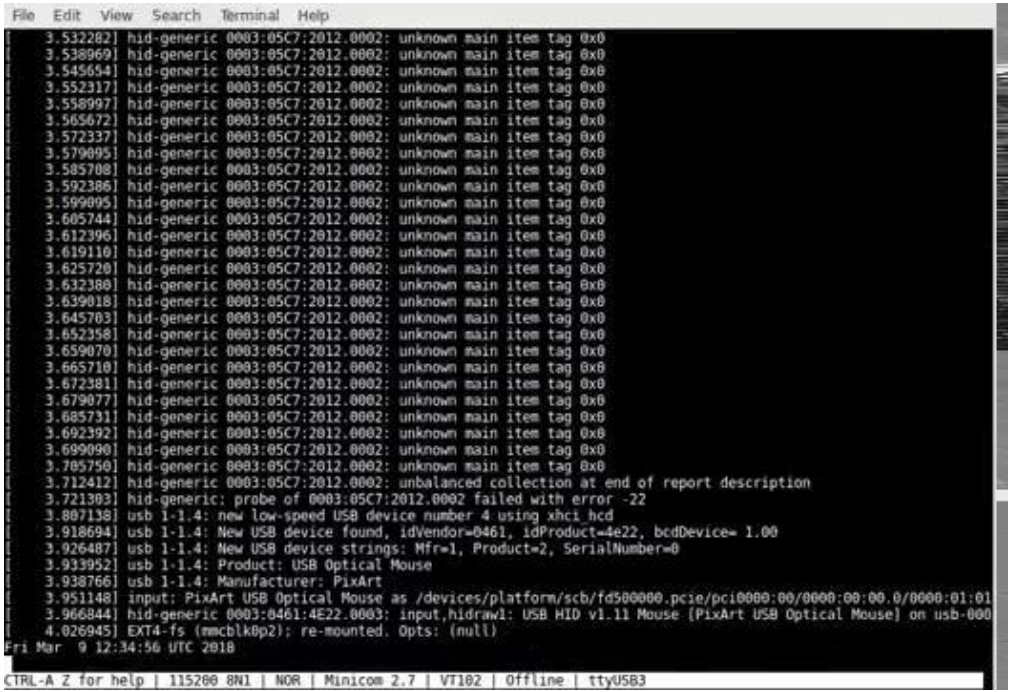
This one: **good**, you want it: that's **ground**: connect it to the black single wire on the USB serial thing

These two: **also good**: **RX** and **TX**
There's probably some way to get them the right way around, but: **don't use the red wire** off the USB thing, and then you can just try the other two either way and one way will work: **bingo!**

there are loads more pins down here too,
I just didn't draw them all

Xen Hypervisor on Raspberry Pi 4: serial

On a nearby machine with the USB serial device plugged in: `minicom /dev/ttyUSB3`



Xen for the Raspberry Pi 4 : simple Xen commands

Testing basic Xen functionality at the console:

- xl list - list running VMs
- xl info - show data about the current hypervisor
- ls -l /dev/xen - examine Linux's Xen device nodes
- xenstore-ls - read the contents of the XenStore tree
- dmesg | grep Xen - see the Linux kernel messages relating to Xen
- xl dmesg - see the Xen boot messages

All these should be familiar if you've used Xen on other systems - and now available on the Raspberry Pi 4!

Xen for the Raspberry Pi 4 : building a guest VM

Let’s boot Yocto Linux inside Yocto Linux! First, build the guest filesystem image:

- **bitbake xen-guest-image-minimal**

Install the pieces needed to run a guest into the running Domain-0 of the Pi:

- Copy in the built **guest filesystem** in a file:
  - .../work/raspberrypi4_64-poky-linux/xen-guest-image-minimal/*/deploy-xen-guest-image-minimal-image-complete/xen-guest-image-minimal-raspberrypi4-64.ext3
    - to: /home/root/xen-guest-image-minimal-raspberrypi4-64.ext3
- Copy in the **guest kernel** file: Image
  - .../work/raspberrypi4_64-poky-linux/linux-raspberrypi/*/deploy-linux-raspberrypi/Image
    - to: /home/root/Image
- Create a new file: **guest.cfg**
  - kernel = "/home/root/Image"
    cmdline = "console=hvc0 earlyprintk=xen sync_console root=/dev/xvda"
    memory = "256"
    name = "rpi4-xen-guest"
    vcpus = 1
    serial="pty"
    disk = ['phy:/dev/loop0,xvda,w']
    vif=['mac=00:11:22:66:88:22,bridge=xenbr0,type=netfront',]

Xen for the Raspberry Pi 4 : prepare for a guest VM

Networking - so the guest can get its own network access:

- Create a bridge and move the eth0 physical device onto it:
  - killall -SIGUSR2 udhcpc # release your existing DHCP lease
  - brctl addbr xenbr0 # create a new bridge called “xenbr0”
  - brctl addif xenbr0 eth0 # put eth0 onto xenbr0
  - killall udhcpc # terminate the DHCP client daemon
  - udhcpc -R -b -p /var/run/udhcpc.xenbr0.pid -i xenbr0 # restart the DHCP client daemon on the new bridge

Disk:

- Loopback mount the ext3 guest filesystem file to make it available as a device
  - losetup /dev/loop0 /home/root/xen-guest-image-minimal-raspberrypi4-64.ext3

Xen for the Raspberry Pi 4 : running a guest VM

xl create -c guest.cfg

Xen for the Raspberry Pi 4 : the current patch series

OK, so what's in this new meta-virtualization patch series to make this work?

- A new "dynamic layer" for *settings that are specific to Xen-on-Raspberry-Pi-4*
  - A series of Linux kernel patches: DMA fixes from the Xen ARM maintainer
  - Enabling the Yocto kernel cache with the Raspberry Pi kernel to build with meta-virtualization
  - Enabling the hardware interrupt controller that Xen needs in the rpi-config
  - Custom Xen-specific bootloader script:
    - Loads the Xen hypervisor binary
    - Loads the Linux kernel
    - Amends the device tree, that the Raspberry Pi has already processed before u-boot
    - Sets the Dom0 kernel command line, to include Xen config settings eg. for the console
  - A Xen hypervisor "defconfig", with settings specific for the Raspberry Pi 4 hardware
  - A Xen-specific SD-card class, to include the Xen binary on the first partition of the image
- A new ARM assembly Xen patch to implement an atomic primitive for spinlocks with the latest gcc in Yocto
- Xen version upgrade to 4.13 (and 4.14 should be coming soon)
- A new method of engaging Xen-specific config settings when DISTRO_FEATURES includes 'xen'

Xen in meta-virtualization: beyond the Pi!

Other example platforms for running **Xen** with Yocto, using **meta-virtualization**:

- **Intel x86-64**: ubiquitous!
  - meta-virtualization has wic tool integration
    - enables simple production of a bootable image:
      - `wic create directdisk-xen -e xen-image-minimal`
      - `dd` the output file to your hard disk and boot it!

- **PCEngines APU2**: low power, low cost (~$100), very open hardware
  - open hardware schematics, has coreboot support
  - add `meta-pcengines` and set `MACHINE = "pcengines-apu2"`
    then use the wic image - `dd` it to a drive and boot into Xen
  - hardware supports D-RTM - that's a big deal: see the OpenXT community for more!

- **runqemu** !
  - `runqemu xen-image-minimal nographic slirp`
    - launch Xen at your command prompt!
      - nb: is currently pretty experimental - has worked with MACHINE = "genericx86-64"; may need some work.

behind the scenes: kas and multiconfig

kas.yaml (1 of 6)

```
# Every file needs to contain a header, that provides kas with information
# about the context of this file.
header:
  # The `version` entry in the header describes for which configuration
  # format version this file was created. It is used by kas to figure
  # out if it is compatible with this file. The version is an integer that
  # is increased on every format change.
  version: 8
# The machine as it is written into the `local.conf` of bitbake.
machine: qemux86-64

# core-image-mimnimal is too limiting
target: core-image-base

# The distro name as it is written into the `local.conf` of bitbake.
distro: poky
```

kas.yaml (2 of 6)

```
repos:
  # Include local layer (e.g. conf/layer.conf)
  meta-custom:

  # Here we include a list of layers from the poky repository to add
  # to bblayers.conf:
  poky:
    url: "https://git.yoctoproject.org/git/poky"
    refspec: master
    layers:
      meta:
      meta-poky:
      meta-yocto-bsp:
```

kas.yaml (3 of 6)

```
  meta-intel:
    url: "https://git.yoctoproject.org/git/meta-intel"
    refspec: master

  meta-openembedded:
    url: "https://git.openembedded.org/meta-openembedded"
    refspec: master
    layers:
      meta-oe:
      meta-filesystems:
      meta-networking:
      meta-perl:
      meta-python:
```

kas.yaml (4 of 6)

```
  meta-acrn:
    url: "https://github.com/intel/meta-acrn.git"
    refspec: master

  meta-virtualization:
    url: "https://git.yoctoproject.org/git/meta-virtualization"
    refspec: master

  meta-cloud-services:
    url: "https://git.yoctoproject.org/git/meta-cloud-services"
    refspec: master

  meta-security:
    url: "https://git.yoctoproject.org/git/meta-security"
    refspec: master
```

kas.yaml (5 of 6)

```
  meta-raspberrypi:
    url: "https://git.yoctoproject.org/git/meta-raspberrypi"
    refspec: master

bblayers_conf_header:
  standard: |
    POKY_BBLAYERS_CONF_VERSION = "2"
    BBPATH = "${TOPDIR}"
    BBFILES ?= ""

local_conf_header:
  standard: |
    CONF_VERSION = "1"

  debug-tweaks: |
    EXTRA_IMAGE_FEATURES = "debug-tweaks"
```

kas.yaml (6 of 6)

```
diskmon: |
  BB_DISKMON_DIRS = "\
      STOPTASKS,${TMPDIR},1G,100K \
      STOPTASKS,${DL_DIR},1G,100K \
      STOPTASKS,${SSTATE_DIR},1G,100K \
      STOPTASKS,/tmp,100M,100K \
      ABORT,${TMPDIR},100M,1K \
      ABORT,${DL_DIR},100M,1K \
      ABORT,${SSTATE_DIR},100M,1K \
      ABORT,/tmp,10M,1K"

distro_features: |
  DISTRO_FEATURES_append = " virtualization"
```

kas-xen-raspberrypi4-64.yaml (1 of 2)

```
header:
  version: 2
  includes:
    - kas.yml

machine: raspberrypi4-64

target:
   - multiconfig:xen-guest:xen-guest-image-minimal
   - xen-image-minimal
local_conf_header:
  tmp_xen_host: |
    TMPDIR = "${TOPDIR}/tmp-xen-host"
  image_fstypes: |
    IMAGE_FSTYPES_append = " wic wic.bmap"
```

kas-xen-raspberrypi4-64.yaml (2 of 2)

```
  xen_distro_features: |
    DISTRO_FEATURES_append = " xen"
  bb_number_threads: |
    BB_NUMBER_THREADS ?= "8"
  parallel_make: |
    PARALLEL_MAKE ?= "-j 8"
  qemu_targets: |
    QEMU_TARGETS = "i386 x86_64 aarch64 arm"
  package_config_remove_sdl: |
    PACKAGECONFIG_remove_pn-qemu += " sdl"
  The packages-from-images recipe needs to know where images are: |
    VESSEL_PACKAGE_DEPLOY_DIR = "${TOPDIR}/tmp-xen-guest/deploy/images/${MACHINE}"
    VESSEL_PACKAGE_MC = "xen-guest"
  Add xen-guest-image-minal-package to xen-host core-image-base: |
    IMAGE_INSTALL_append_pn-xen-image-minimal = " xen-guest-image-minimal-package"
```

conf/muticonfig/xen-guest.conf


```
DISTRO = "poky"
TMPDIR = "${TOPDIR}/tmp-xen-guest"
PREFERRED_PROVIDER_virtual/kernel = "linux-yocto"
IMAGE_FSTYPES_append = " wic wic.qcow2"
```

xen-guest-package.bbclass (1 of 2)


```
SUMMARY = "Package for ${IMAGE_NAME}"
# This license statement is a lie. Ideally set it to something more
appropriate.
LICENSE = "CLOSED"

PACKAGE_ARCH = "${MACHINE_ARCH}"
PACKAGES = "${PN}"

INHIBIT_DEFAULT_DEPS = "1"

# Variables to control where images are found: the multiconfig name,
and the deploy dir.
VESSEL_PACKAGE_MC ?= ""
VESSEL_PACKAGE_DEPLOY_DIR ?= "${DEPLOY_DIR_IMAGE}"
```

xen-guest-package.bbclass (1 of 2)


```
# The name of the image
IMAGE_NAME := "${@d.getVar('PN').replace('-package', '')}"
# Where to install the image
vesseldir ?= "${localstatedir}/lib/machines"

do_install[depends] += "virtual/kernel:do_deploy"
do_install[mcdepends] +=
"multiconfig::${VESSEL_PACKAGE_MC}:${IMAGE_NAME}:do_image_complete"

do_install () {
    install -d ${D}${vesseldir}
    install ${VESSEL_PACKAGE_DEPLOY_DIR}/Image ${D}${vesseldir}/Image
    install ${VESSEL_PACKAGE_DEPLOY_DIR}/${IMAGE_NAME}-${MACHINE}.wic.qcow2
${D}${vesseldir}/${IMAGE_NAME}.wic.qcow2
}
```

xen-guest-image-minimal-package.bb


```
SUMMARY = "Package wrapper around a minimal Xen guest image"
# THis is a lie, see the license manifest inside the xen guest image
LICENSE="CLOSED"

inherit xen-guest-package

SRC_URI = "file://xen-guest-image-minimal.cfg"

do_install_append() {
  install ${S}/../xen-guest-image-minimal.cfg ${D}${vesseldir}/
}
```

xen-guest-image-minimal-package.cfg


```
kernel ="/var/lib/machines/Image"
cmdline ="console=hvc0 earlyprintk=xen sync_console root=/dev/xvda"
memory = “256”
name = "rpi4-xen-guest"
vcpus = 1
serial = "pty"
disk = [“format=qcow2,vdev=xvda,access=rw,\
        target=/var/lib/machines/xen-guest-image-minimal.wic.qcow2”]
vif = ['mac:00.11.22.66.88.22,bridge=xenbr0,type=netfront']
```

Thanks

- Xen Community
  - for the Xen hypervisor and Linux kernel work to make this possible
    - Stefano Stabellini @ Xilinx, Julien Grall @ Amazon
  - for the interest in the Raspberry Pi 4
    - Roman Shaposhnik @ Eve Project, Zededa
  - hey! See you at the Xen Design and Developer Summit next week!
- Yocto and OpenEmbedded meta-virtualization Community
  - for the first Xen on Raspberry Pi 4 meta-virtualization patch submissions
    - Corey Minyard @ MontaVista, Stewart Hildebrand @ DornerWorks
  - for the support for bringing Xen work in, in a maintainable way
    - Bruce Ashfield @ Xilinx, Bertrand Marquis @ ARM
  - for this talk opportunity!
    - Tim Orling @ Yocto Project, Intel, David Reyna @ Wind River
- OpenXT Community
  - for supporting my involvement with Xen and OpenEmbedded
- Raspberry Pi Community
  - for developing and promoting accessible hardware with Open Source software

KVM Hypervisor

The Linux Kernel Hypervisor

https://www.linux-kvm.org/page/Main_Page

Kernel-based Virtual Machine (KVM)

”The Kernel-based virtual Machine (KVM) is a full virtualization hypervisor for Linux. The work of the **KVM hypervisor** is handled by the **Linux kernel**. Each guest in KVM runs as a process and can be managed by Linux tools such as **top** and **kill**.

KVM isn’t a complete virtualization solution. It depends on both the **libvirt** tools for management and the open source processor emulator **QEMU** for hardware emulation. Therefore, you will need those **installed** as well.”

-- Stephen Figgins, Robert Love, Arnold Robbins, Ellen Siever,
Linux in a Nutshell, 6th ed., O’Reilly Media, Inc, 2009.

Let’s do this!

packagegroup-kvm-host.bb

```
SUMMARY = "Provides a set of tools for hosting KVM guests."

inherit packagegroup

#TODO: cloud-init

RDEPENDS_${PN} = "\
    packagegroup-core-boot \
    qemu \
    libvirt \
    libvirt-libvirtd \
    libvirt-virsh \
        "
```

kvm-binary-image-vessel-package.bbclass

(Avoiding the overloaded meaning of “container”)

```
SUMMARY = "Package for ${IMAGE_NAME}"
# This license statement is a lie. Ideally set it to something more appropriate.
LICENSE = "CLOSED"

INHIBIT_DEFAULT_DEPS = "1"

inherit bin_package

# Where to install the image
vesseldir ?= "${localstatedir}/lib/libvirt/images"

do_install[depends] += "libvirt:do_install"

do_install () {
        install -d ${D}${vesseldir}
        install ${S}/../${VESSEL_PAYLOAD_NAME} ${D}${vesseldir}/${VESSEL_PAYLOAD_NAME}
}
```

Based on https://github.com/intel/meta-acrn/blob/master/classes/container-package.bbclass by Ross Burton

ubuntu-kvm-image-package.bb

```
SUMMARY = "Ubuntu cloud kvm image"
# Probably this should be Canonical IPRights?
LICENSE="CLOSED"

inherit kvm-binary-guest-package

# precise, xenial and bionic do not have kvm images
# eoan, focal and groovy do
UBUNTU_BASE_URL ??= "https://cloud-images.ubuntu.com"
UBUNTU_RELEASE ??="focal"
UBUNTU_IMAGE_ARCH ??="amd64"
UBUNTU_IMAGE_NAME ?=
"${UBUNTU_RELEASE}-server-cloudimg-${UBUNTU_IMAGE_ARCH}-disk-kvm.img"
UBUNTU_IMAGE_DATE ?= "current"
VESSEL_PAYLOAD_NAME = "${UBUNTU_IMAGE_NAME}”

[…]
```

ubuntu-kvm-image-package.bb (cont'd)

```
SRC_URI =
"${UBUNTU_BASE_URL}/${UBUNTU_RELEASE}/${UBUNTU_IMAGE_DATE}/${UBUNTU_IMAGE_NAME}"
SHA256SUMS_URI = "${UBUNTU_BASE_URL}/${UBUNTU_RELEASE}/${UBUNTU_IMAGE_DATE}/SHA256SUMS"

# See  http://www.burtonini.com/blog/2017/06/13/dynamic-source-checksums
do_fetch[prefuncs]  += "fetch_checksums"

python fetch_checksums() {
    import re
    import urllib

    match = "*{}".format(d.getVar("UBUNTU_IMAGE_NAME"))
    for line in urllib.request.urlopen(d.getVar("SHA256SUMS_URI")):
        (sha256, filename) = line.decode("ascii").strip().split()
        if filename == match:
            d.setVarFlag("SRC_URI", "sha256sum", sha256)
            return
    bb.error("Could not find remote checksum for %s" %  filename)
}
```

launch-ubuntu-kvm.sh

```
qemu-system-x86_64 -m 4096 -drive
format=qcow2,file=/var/lib/libvirt/images/focal-server-cloudimg-amd64-disk-kvm.img
-nographic
```

Unmodified Ubuntu 20.04 cloud image

Future work

- Secure Boot (see Bonus)
- cloud-init (username and password)
- virsh (use a template to create XML)
- Launch script to automatically boot the guest
- Insert ssh keys into guest (cloud-init?)

ACRN™ Hypervisor

ACRN Type 1 Hypervisor

ACRN™ is a, flexible, lightweight reference hypervisor, built with real-time and safety-criticality in mind, and optimized to streamline embedded development through an open source platform. ACRN defines a device hypervisor reference stack and an architecture for running multiple software subsystems, managed securely, on a consolidated system by means of a virtual machine manager (VMM). It also defines a reference framework implementation for virtual device emulation, called the “ACRN Device Model”.

The **ACRN Hypervisor is a Type 1** reference **hypervisor** stack, **running directly** on the **bare-metal hardware**, and is suitable for a variety of IoT and embedded device solutions. The ACRN hypervisor addresses the gap that currently exists between datacenter hypervisors, and hard partitioning hypervisors. The ACRN hypervisor architecture partitions the system into different functional domains, with carefully selected user VM sharing optimizations for IoT and embedded devices.

https://projectacrn.github.io/latest/introduction/index.html

Great. How do I use it?

(With Yocto Project!)

https://github.com/intel/meta-acrn/blob/master/docs/getting-started.md

conf/local.conf

```
MACHINE = "intel-corei7-64"
TMPDIR = "${TOPDIR}/tmp-acrn-sos"
DISTRO = "acrn-demo-sos"
# Also use the 'uos' configuration
BBMULTICONFIG = "uos"
# The packages-from-images class (container-package.bbclass) needs to know where images
are
CONTAINER_PACKAGE_DEPLOY_DIR = "${TOPDIR}/master-acrn-uos/deploy/images/${MACHINE}"
CONTAINER_PACKAGE_MC = "uos"
# Add core-image-base-package to acrn-image-base
IMAGE_INSTALL_append_pn-acrn-image-base = " core-image-base-package"
# Add core-image-weston-package to acrn-image-sato
IMAGE_INSTALL_append_pn-acrn-image-sato = " core-image-weston-package"
# set preferred kernel for sos
PREFERRED_PROVIDER_virtual/kernel = "linux-intel-acrn-sos”
# libvirt in meta-virtualization conflicts
PREFERRED_RPROVIDER_libvirt = "acrn-libvirt"
PREFERRED_RPROVIDER_libvirt-virsh = "acrn-libvirt"
PREFERRED_RPROVIDER_libvirt-libvirtd = "acrn-libvirt"
```

Add layers

```
$ bitbake-layers add-layer meta-intel
$ bitbake-layers add-layer meta-acrn
```

conf/multiconfig/uos.conf

```
DISTRO = "acrn-demo-uos"
TMPDIR = "${TOPDIR}/tmp-acrn-uos"
PREFERRED_PROVIDER_virtual/kernel = "linux-intel-acrn-uos"
```

Future Update: secureboot dm-verity

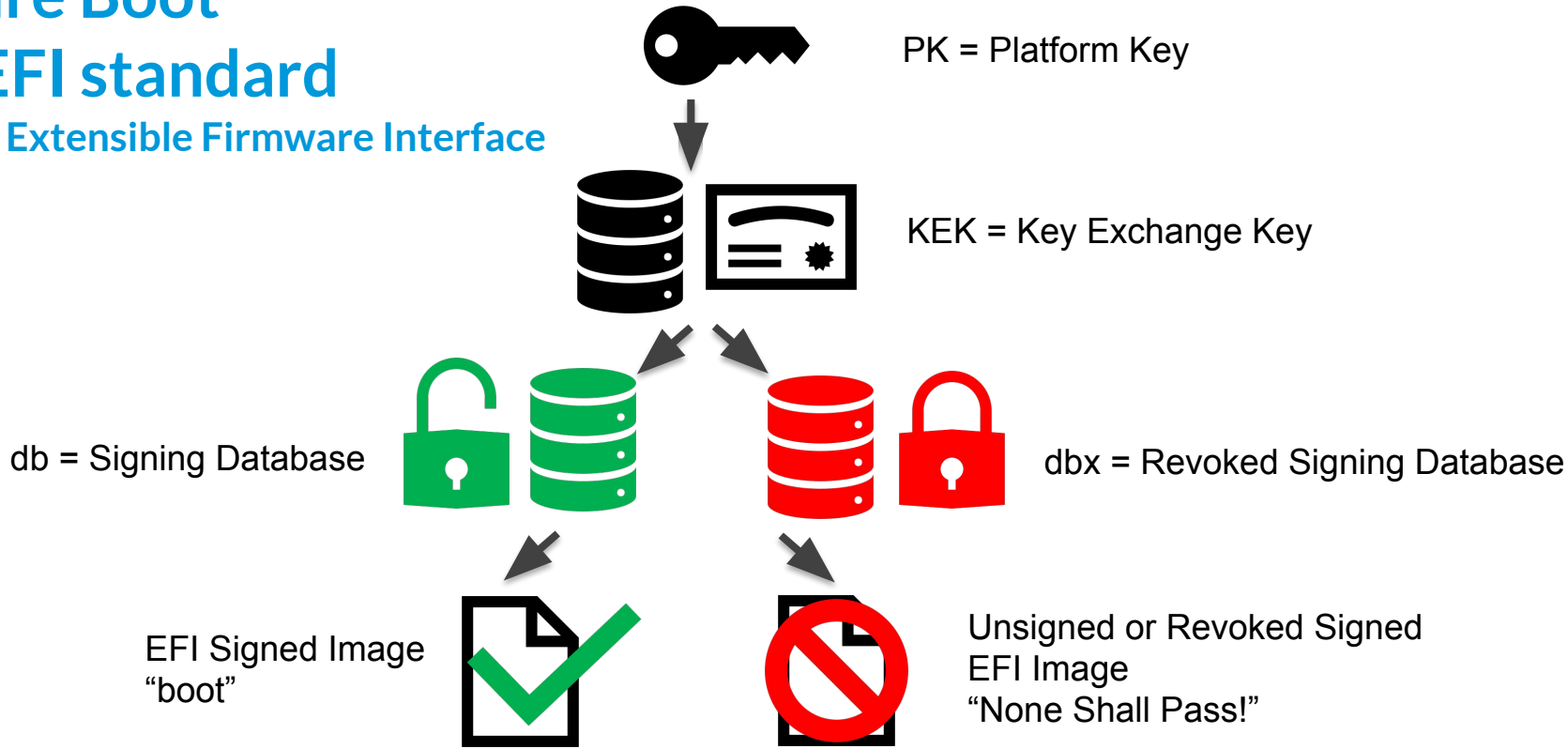
Secure Boot

A Chain of Trust

What is this “Secure Boot” you speak of?

(And why should I trust you?)

Secure Boot
A UEFI standard
Unified Extensible Firmware Interface



It’s just efivars.

(KISS)

/sys/firmware/efi/efivars

EFI Kernel Config options

```
# SPDX-License-Identifier: MIT
# EFI Support

# Dependencies
CONFIG_PCI=y
CONFIG_ACPI=y

# Enable basic EFI support
CONFIG_EFI=y
CONFIG_EFI_STUB=y
CONFIG_EFIVAR_FS=y
CONFIG_EFI_MIXED=y
```

http://git.yoctoproject.org/cgit/cgit.cgi/yocto-kernel-cache/tree/cfg/efi.cfg

http://git.yoctoproject.org/cgit/cgit.cgi/yocto-kernel-cache/tree/cfg/efi-ext.cfg

https://wiki.archlinux.org/index.php/Unified_Extensible_Firmware_Interface

EFI Kernel Config options

```
# Extended EFI support
# Dependencies
# efi.cfg
CONFIG_FB=y
CONFIG_VT=y
CONFIG_BLOCK=y
CONFIG_PARTITION_ADVANCED=y
# Add support for optional EFI features
CONFIG_FRAMEBUFFER_CONSOLE=y
CONFIG_FB_EFI=y
CONFIG_EFI_VARS=m
CONFIG_EFI_PARTITION=y
```

http://git.yoctoproject.org/cgit/cgit.cgi/yocto-kernel-cache/tree/cfg/efi.cfg

http://git.yoctoproject.org/cgit/cgit.cgi/yocto-kernel-cache/tree/cfg/efi-ext.cfg

https://wiki.archlinux.org/index.php/Unified_Extensible_Firmware_Interface

Take away this one thing.

https://wiki.gentoo.org/wiki/Sakaki%27s_EFI_Install_Guide/Configuring_Secure_Boot

TL;DR

Tell me how to boot my self-signed images!

Self-signed images: Backup your device keys

```
root@intel-corei7-64 ~ #mkdir -p -v /etc/efikeys
root@intel-corei7-64 ~ #chmod -v 700 /etc/efikeys
root@intel-corei7-64 ~ #cd /etc/efikeys
root@intel-corei7-64 /etc/efikeys #efi-readvar -v PK -o old_PK.esl
root@intel-corei7-64 /etc/efikeys #efi-readvar -v KEK -o old_KEK.esl
root@intel-corei7-64 /etc/efikeys #efi-readvar -v db -o old_db.esl
root@intel-corei7-64 /etc/efikeys #efi-readvar -v dbx -o old_dbx.esl
```

Self-signed images: Generate your signing keys

```
root@intel-corei7-64 /etc/efikeys #openssl req -new -x509 -newkey
rsa:2048 -subj "/CN=secret platform key/" -keyout PK.key -out
PK.crt -days 3650 -nodes -sha256
root@intel-corei7-64  /etc/efikeys #openssl req -new -x509 -newkey
rsa:2048 -subj "/CN=secret key-exchange-key/" -keyout KEK.key -out
KEK.crt -days 3650 -nodes -sha256
root@intel-corei7-64  /etc/efikeys #openssl req -new -x509 -newkey
rsa:2048 -subj "/CN=secret kernel-signing key/" -keyout db.key -out
db.crt -days 3650 -nodes -sha256
```

And protect them

```
root@intel-corei7-64 /etc/efikeys #chmod -v 400 *.key
```

Self-signed images: Preparing Keystore Update Files from Keys

```
root@intel-corei7-64 /etc/efikeys #cert-to-efi-sig-list -g "$(uuidgen)"
PK.crt PK.esl
root@intel-corei7-64 /etc/efikeys #sign-efi-sig-list  -k PK.key -c PK.crt  PK
PK.esl PK.auth

root@intel-corei7-64 /etc/efikeys #cert-to-efi-sig-list -g "$(uuidgen)"
KEK.crt KEK.esl
root@intel-corei7-64 /etc/efikeys #sign-efi-sig-list -a  -k PK.key -c PK.crt
KEK KEK.esl KEK.auth

root@intel-corei7-64 /etc/efikeys #cert-to-efi-sig-list -g "$(uuidgen)"
db.crt db.esl
root@intel-corei7-64 /etc/efikeys #sign-efi-sig-list -a  -k KEK.key -c
KEK.crt db db.esl db.auth

root@intel-corei7-64 /etc/efikeys #sign-efi-sig-list  -k KEK.key -c KEK.crt
dbx old_dbx.esl old_dbx.auth
```

Self-signed images: Write the vars

...

Bonus

How to fix GPT errors on flashed USB or SD card

How to fix GPT errors on flashed USB

- Do warnings bother you?
- Especially in dmesg?

```
[1875147.012246] GPT:Primary header thinks Alt. header is not at the end of the disk.
[1875147.012248] GPT:7079479 != 61341695
[1875147.012249] GPT:Alternate GPT header not at the end of the disk.
[1875147.012250] GPT:7079479 != 61341695
[1875147.012250] GPT: Use GNU Parted to correct GPT errors.
[1875147.012259]  sdd: sdd1 sdd2 sdd3
```

gdisk to the rescue

Be sure to use the correct device

NOTE:
Use sgdisk for pure command line

```
$ sudo gdisk /dev/sdd
GPT fdisk (gdisk) version 1.0.3

Partition table scan:
  MBR: protective
  BSD: not present
  APM: not present
  GPT: present

Found valid GPT with protective MBR; using GPT.

Command (? for help): v

Problem: The secondary header's self-pointer indicates that it doesn't reside
at the end of the disk. If you've added a disk to a RAID array, use the 'e'
option on the experts' menu to adjust the secondary header's and partition
table's locations.

Identified 1 problems!
```

```
x       extra functionality (experts only)
Command (? for help): x

e       relocate backup data structures to the end of the disk
Expert command (? for help): e
Relocating backup data structures to the end of the disk

v       verify disk
Expert command (? for help): v

No problems found. 54264271 free sectors (25.9 GiB) available in 3
segments, the largest of which is 54262217 (25.9 GiB) in size.

w       write table to disk and exit
Expert command (? for help): w

Final checks complete. About to write GPT data. THIS WILL OVERWRITE EXISTING
PARTITIONS!!

Do you want to proceed? (Y/N): y
OK; writing new GUID partition table (GPT) to /dev/sdd.
The operation has completed successfully.
```

What is the Yocto Project® ?

IT'S NOT AN EMBEDDED LINUX DISTRIBUTION, IT CREATES A CUSTOM ONE FOR YOU.

The Yocto Project (YP) is an open source collaboration project that helps developers create custom Linux-based systems regardless of the hardware architecture.

The project provides a flexible set of tools and a space where embedded developers worldwide can share technologies, software stacks, configurations, and best practices that can be used to create tailored Linux images for embedded and IOT devices, or anywhere a customized Linux OS is needed.

yocto PROJECT | THE LINUX FOUNDATION